Эффективное программирование современных микропроцессоров и мультипроцессоров

Практическое задание 2

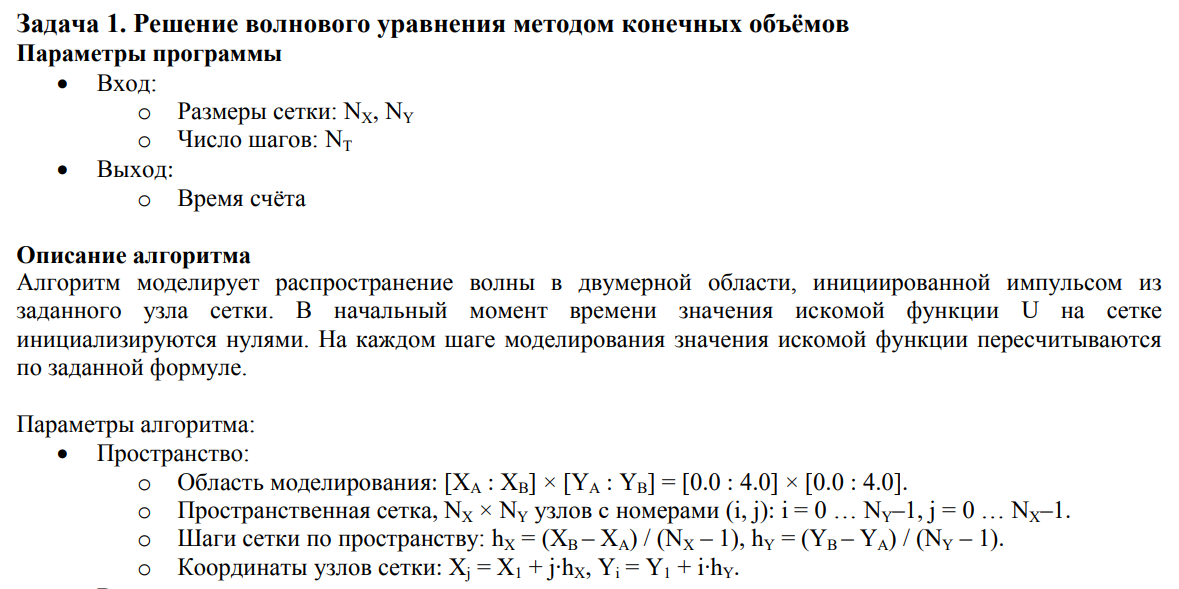
Выполнила:

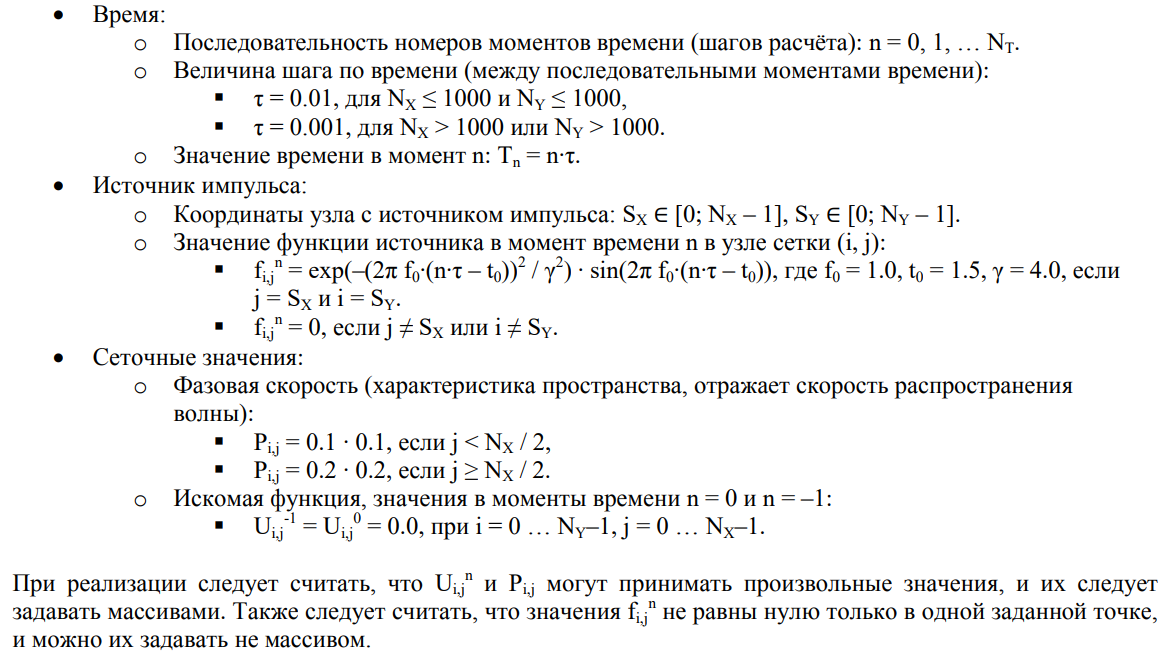
…

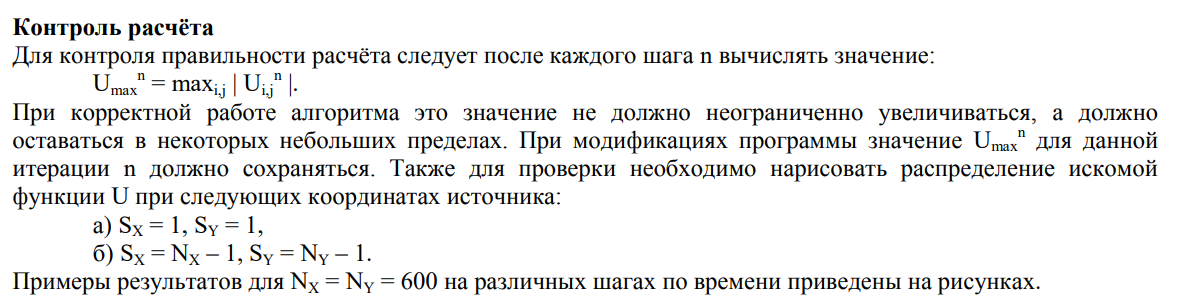
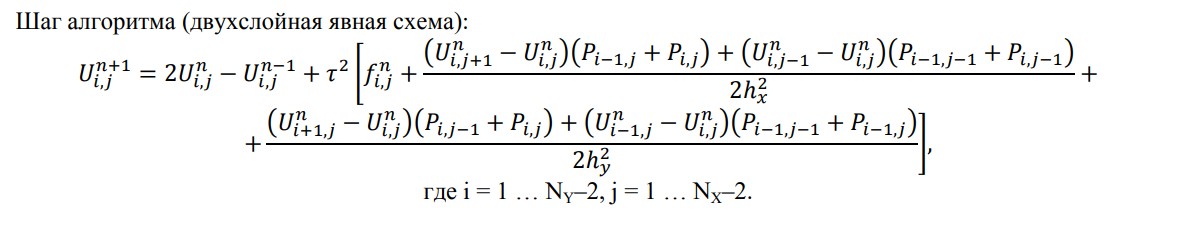
Группа …

Преподаватель:

…







**Результаты тестирования различных версий программы**

Nx = Ny = 8000, Nt = 100;

1. Изначальный вариант программы

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 204.2 | 47.2 | 37.5 | 22.9 |

1. Исключение повторяющихся вычислений

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 203.0 | 42.6 | 34.6 | 22.3 |

1. Замена всех операций деления на умножение

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 195.2 | 42.0 | 33.4 | 21.9 |

1. Заблаговременное обращение к массиву

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 139.7 | 38.7 | 32.9 | 22.4 |

1. Заблаговременное вычисление индекса элемента массива

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 102.8 | 31.5 | 24.5 | 23.7 |

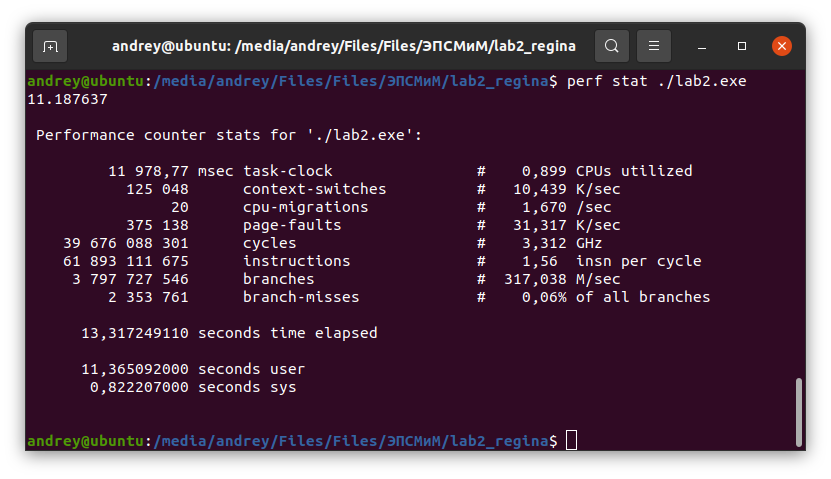
1. Векторизация

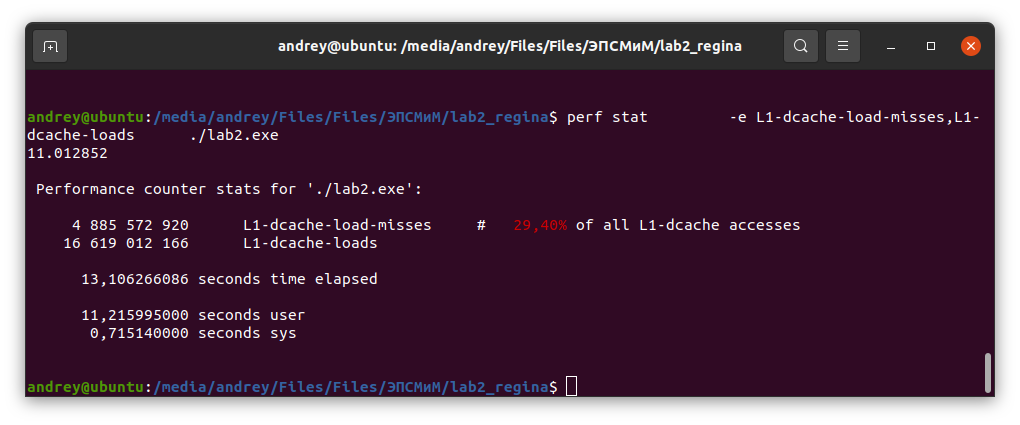
|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 101.2 | 18.2 | 15.9 | 15.4 |

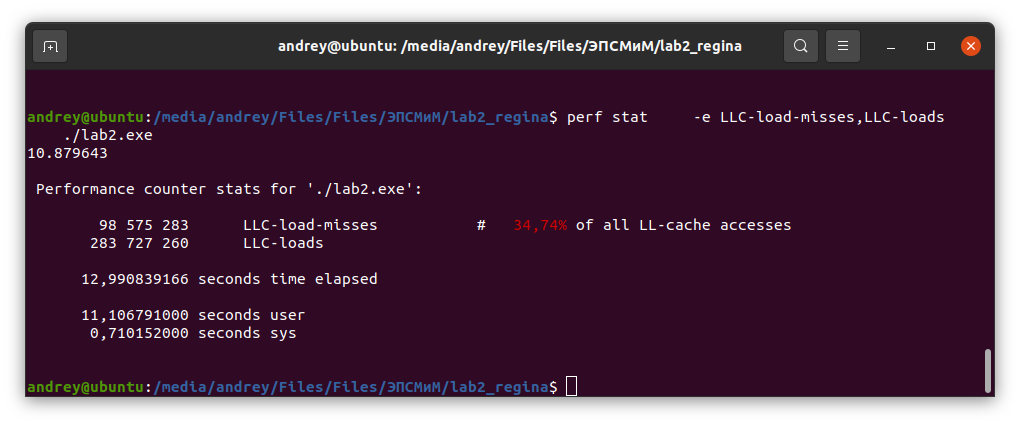
**Вывод**

Ручная векторизация вычислений с помощью Intel vector intrinsics уменьшила время работы программы в 1.5 раза. Векторизация позволила выполнять однотипные операции одновременно, но привела к увеличению числа обращений к памяти (так как при последовательном считывании векторов из четырёх элементов с шагом в один элемент, часть элементов считывается не единожды) и увеличению числа кэш-промахов (увеличилась скорость вычислений, и аппаратная предвыборка может не успевать подавать данные в кэш). Это привело к более равномерному распределению нагрузки между работой памяти и вычислениями.

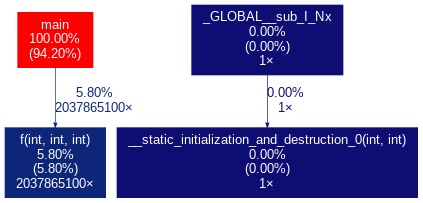
**Профилирование perf**



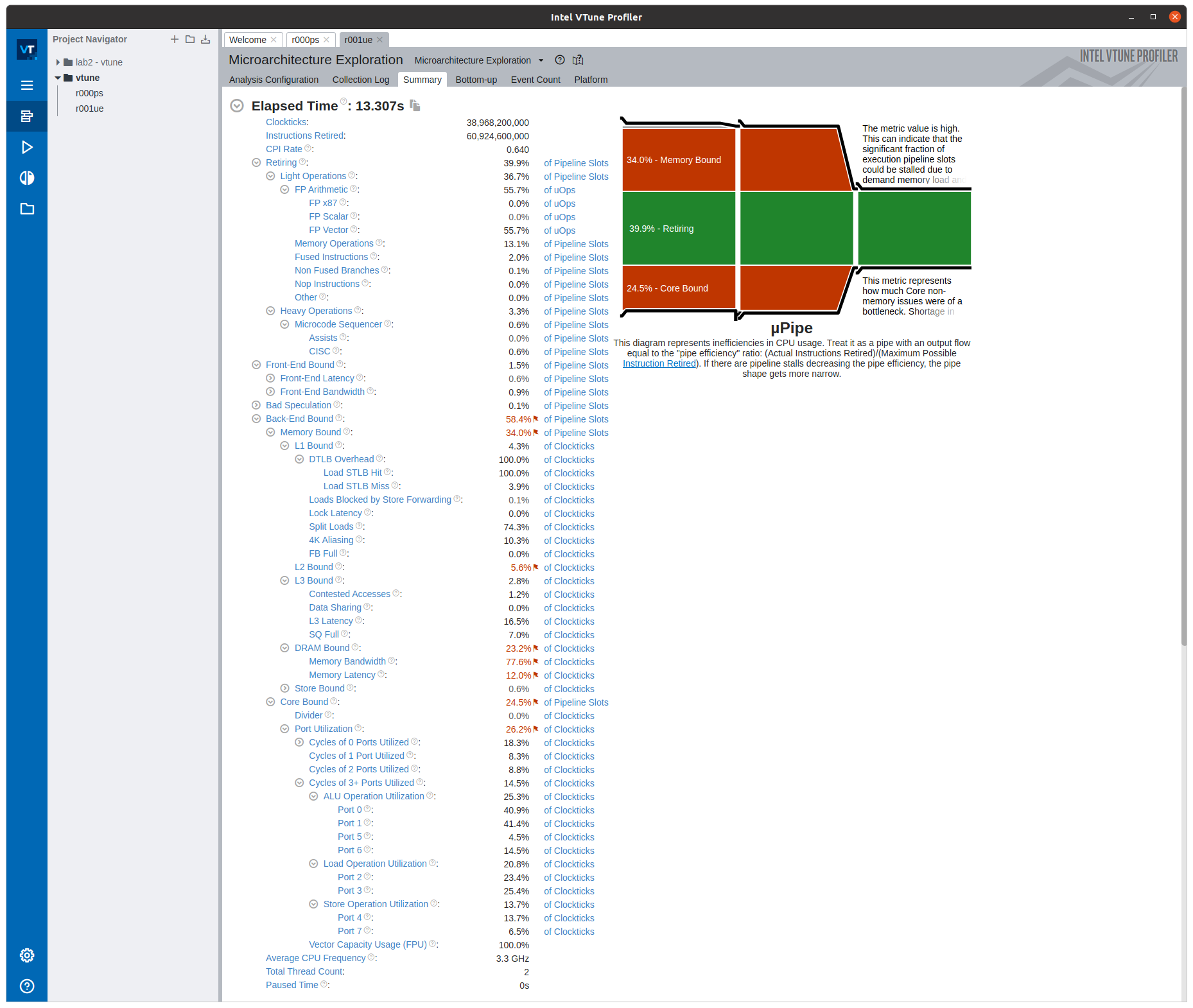




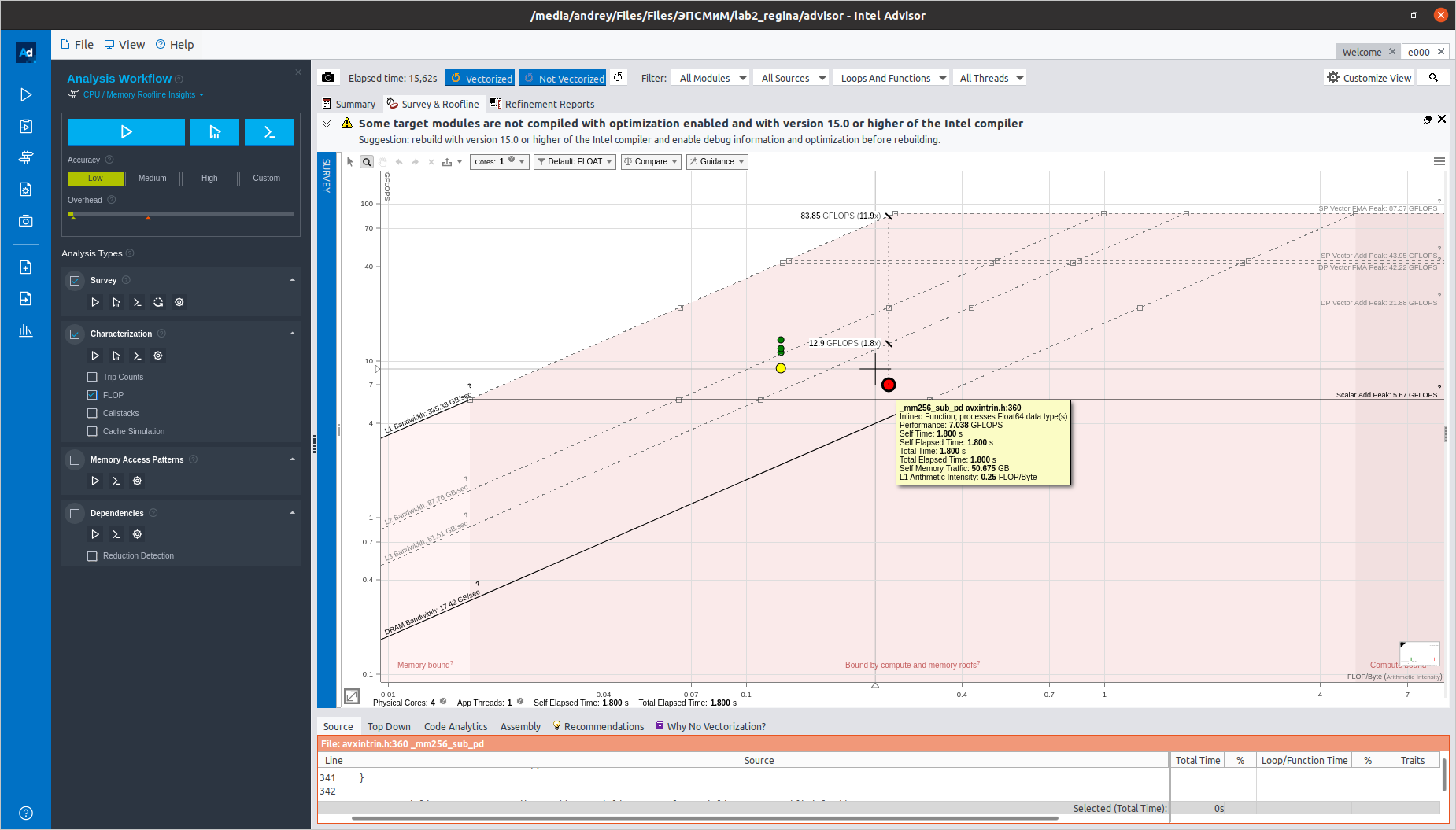
**Граф вызовов**



**Профилирование Vtune**



**Roofline-модель**



#include <iostream>

#include <math.h>

#include <cstdlib>

#include <ctime>

#include <stdio.h>

#include <cstring>

#include <memory>

#include <xmmintrin.h>

#include <immintrin.h>

using namespace std;

int Nx = 8000, Ny = 8000, Nt = 100;

int Sx = 1, Sy = 1;

double t;

const int ALIGN\_INDENT = 64;

const int VECTOR\_SIZE = 4;

double f (int n) {

double f0 = 1.0, t0 = 1.5, c = 4.0;

double arg = 2\*3.14\*f0\*(n\*t - t0);

return exp(-0.0625 \* (arg\*arg)) \* sin(arg);

}

int main(int argc, char\*\* argv) {

double Xa = 0.0, Ya = 0.0, Xb = 4.0, Yb = 4.0;

double hx = (Xb - Xa) / (Nx - 1), hy = (Yb - Ya) / (Ny - 1);

if (Nx <= 1000 && Ny <= 1000) t = 0.01;

else t = 0.001;

double\* P = (double\*)aligned\_alloc(ALIGN\_INDENT, Nx \* Ny \* sizeof(double));

for (int i = 0; i < Ny; i++)

for (int j = 0; j < Nx; j++) {

if (j < Nx/2) P[i\*Nx + j] = 0.1\*0.1;

else P[i\*Nx + j] = 0.2\*0.2;

}

double\*\* U = (double\*\*)malloc(2 \* sizeof(double\*));

U[0] = (double\*)aligned\_alloc(ALIGN\_INDENT, Nx \* Ny \* sizeof(double));

U[1] = (double\*)aligned\_alloc(ALIGN\_INDENT, Nx \* Ny \* sizeof(double));

for (int i = 0; i < Ny; i++)

for (int j = 0; j < Nx; j++) {

U[0][i\*Nx + j] = 0.0;

U[1][i\*Nx + j] = 0.0;

}

double d\_x = 0.5 / (hx \* hx), d\_y = 0.5 / (hy \* hy), t2 = t \* t;

double t2\_arr[VECTOR\_SIZE] = {t2, t2, t2, t2};

double two\_arr[VECTOR\_SIZE] = {2.0, 2.0, 2.0, 2.0};

double f\_arr[VECTOR\_SIZE] = {0.0, 0.0, 0.0, 0.0};

double d\_x\_arr[VECTOR\_SIZE] = {d\_x, d\_x, d\_x, d\_x};

double d\_y\_arr[VECTOR\_SIZE] = {d\_y, d\_y, d\_y, d\_y};

struct timespec start, end;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

for (int n = 1; n < Nt; n++) {

double\* U\_new = U[n%2];

double\* U\_prv = U[(n+1)%2];

double f\_value = f(n);

for (int i = 1; i < Ny-1; i++) {

int row = i \* Nx, row\_ = (i - 1) \* Nx, row1 = (i + 1) \* Nx;

for (int j = 1; j < Nx-3; j += VECTOR\_SIZE) {

int rowcol = row + j, rowcol\_ = row + j - 1, rowcol1 = row + j + 1,

row\_col = row\_ + j, row1col = row1 + j, row\_col\_ = row\_col - 1;

memset(f\_arr, 0.0, 4 \* sizeof(double));

if (i == Sy && (Sx - j) < VECTOR\_SIZE && (Sx - j) >=0)

f\_arr[(Sx - j)] = f\_value;

\_\_m256d first = \_mm256\_loadu\_pd(U\_prv+rowcol1);

\_\_m256d second = \_mm256\_loadu\_pd(U\_prv+rowcol);

\_\_m256d res1 = \_mm256\_sub\_pd(first, second);

first = \_mm256\_loadu\_pd(P+row\_col);

second = \_mm256\_loadu\_pd(P+rowcol);

\_\_m256d res2 = \_mm256\_add\_pd(first, second);

\_\_m256d firstPath = \_mm256\_mul\_pd(res1, res2);

first = \_mm256\_loadu\_pd(U\_prv+rowcol\_);

second = \_mm256\_loadu\_pd(U\_prv+rowcol);

res1 = \_mm256\_sub\_pd(first, second);

first = \_mm256\_loadu\_pd(P+row\_col\_);

second = \_mm256\_loadu\_pd(P+rowcol\_);

res2 = \_mm256\_add\_pd(first, second);

\_\_m256d secondPath = \_mm256\_mul\_pd(res1, res2);

\_\_m256d lineDividend = \_mm256\_add\_pd(firstPath, secondPath);

first = \_mm256\_loadu\_pd(d\_x\_arr);

\_\_m256d firstLineResult = \_mm256\_mul\_pd(lineDividend, first);

first = \_mm256\_loadu\_pd(U\_prv+row1col);

second = \_mm256\_loadu\_pd(U\_prv+rowcol);

res1 = \_mm256\_sub\_pd(first, second);

first = \_mm256\_loadu\_pd(P+rowcol\_);

second = \_mm256\_loadu\_pd(P+rowcol);

res2 = \_mm256\_add\_pd(first, second);

firstPath = \_mm256\_mul\_pd(res1, res2);

first = \_mm256\_loadu\_pd(U\_prv+row\_col);

second = \_mm256\_loadu\_pd(U\_prv+rowcol);

res1 = \_mm256\_sub\_pd(first, second);

first = \_mm256\_loadu\_pd(P+row\_col\_);

second = \_mm256\_loadu\_pd(P+row\_col);

res2 = \_mm256\_add\_pd(first, second);

secondPath = \_mm256\_mul\_pd(res1, res2);

lineDividend = \_mm256\_add\_pd(firstPath, secondPath);

first = \_mm256\_loadu\_pd(d\_y\_arr);

\_\_m256d secondLineResult = \_mm256\_mul\_pd(lineDividend, first);

\_\_m256d sumOfTwoLines = \_mm256\_add\_pd(firstLineResult, secondLineResult);

\_\_m256d impulseVector = \_mm256\_loadu\_pd(f\_arr);

\_\_m256d bracketResult = \_mm256\_add\_pd(sumOfTwoLines, impulseVector);

first = \_mm256\_loadu\_pd(two\_arr);

second = \_mm256\_loadu\_pd(U\_prv+rowcol);

\_\_m256d totalResult = \_mm256\_mul\_pd(first, second);

second = \_mm256\_loadu\_pd(U\_new+rowcol);

totalResult = \_mm256\_sub\_pd(totalResult, second);

\_\_m256d tauSquareVector = \_mm256\_loadu\_pd(t2\_arr);

\_\_m256d mulResult = \_mm256\_mul\_pd(tauSquareVector, bracketResult);

totalResult = \_mm256\_add\_pd(totalResult, mulResult);

\_mm256\_storeu\_pd(U\_new+rowcol, totalResult);

}

}

}

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

double time = end.tv\_sec-start.tv\_sec + 0.000000001\*(end.tv\_nsec-start.tv\_nsec);

printf("%lf\n", time);

FILE\* file = fopen("file.dat", "wb");

fwrite(U[0], sizeof(double), Nx \* Ny, file);

fclose(file);

return 0;

}